

Graphics-oriented software development tools receive increasing attention in microcontroller land. To many aspiring developers from the industrial automation or high-power electrical engineering disciplines, traditional development tools represent insurmountable hurdles because of their apparent complexity and lack of user-friendly interfaces. In these cases, an all-graphic development system like ST-Realizer may be very welcome. Compared with previous releases of the program, the new 32-bit version of ST-Realizer offers improved real-time abilities, and for the first time allows interrupt-controlled program chunks to be described.

By R. Nandlinger (Stmicroelectronics, Munich)

ST-Realizer®

for ST6 and ST7 microcontrollers

a next-generation
graphics-oriented development tool

Many of you with at least one-time experience in writing microcontroller assembly code will be aware of the problems that may arise during the project. To begin with, the programmer may have to familiarise himself with possibly complex architectures and instruction sets of an unknown microcontroller. Next, the program development begins, followed by commissioning of the application. During this phase, problems often occur owing to insufficient knowledge of idiosyncrasies or controller-specific specialities. It is then hoped that deep study of the datasheets and application notes result in the finding solutions. All of these factors often cause considerable stress during the software development phases. As a result, the programmer runs out of time for the test phase and for writing proper documentation with the new software product. These shortcomings, in turn, lead to modifications at a later time, and, in the worst case, to recalling products already supplied to customers.

The development concept

The graphics-based development software called ST-Realizer developed by the Dutch company **Actum Solutions**



offers a means of solving the above problems in a very efficient way. The idea behind the concept is obvious at the very first step you take to develop a new application.

At the start of a project, most designers have a clear idea about the function of a new piece of equipment. Everything that follows, including the selection of a suitable controller architecture, the size of the controller enclosure,

learning the programming language and debugging and documenting one's progress all seem to get into the way of reaching the ultimate goal, and one would certainly like to shift all side-activities well into the background!

Here, exactly, lies the advantage of graphics-oriented programming. By simply combining logic symbols picked from clearly structured libraries, a few

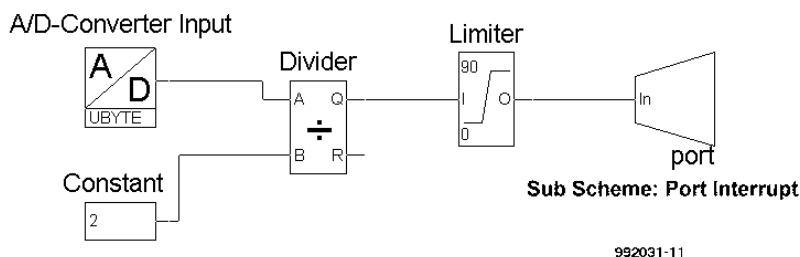


Figure 1. Main program of example version 'A' (dimmer with potentiometer control).

mouse clicks in an Editor-like environment are sufficient to describe (actually, draw) the function of an application. Learning this programming method is highly intuitive because the symbolic language used has been kept very simple. Just as 'pictogram language' used on, say, airport terminals is easily understood by anyone despite his/her mother tongue, the symbols employed by ST-Realizer are heavily based on circuit symbols that should be familiar from electrical engineering. Combining individual symbols largely follows traditional circuit design by simply connecting lines.

Once the program flow has been described, an integrated simulation program may be used to check, and if necessary improve, the results. If the designer is satisfied with the results, he/she picks a suitable type from the list of available microcontrollers, and prompts the integrated analyser to generate the final version of the machine code to be loaded into the target microcontroller. As a useful side product, the designer receives 'free' documentation of his program simply by printing the program flowchart.

The next generation

Based on structured programming, previous versions of the ST-Realizer were mainly aimed at developing processes in which timekeeping was not a critical factor. The new 32-bit version for the first

time allows interrupt-driven routines to be described. This addition has resulted in a plethora of new application options for the user.

An example may help at this point to show the simplicity of it all, even for inexperienced programmers.

Assignment

Your assignment is to develop two versions of a microcontroller-driven dimmer for simple lamps powered from the mains. Version A has to employ a potentiometer as the intensity control. Version B, on the other hand, is to make use of two pushbuttons for the same function.

Functional description

The lamp intensity is governed by controlling the trigger instant, or phase angle, of a triac. With version A, this phase angle is proportional with the analogue value supplied by the potentiometer. With version B, the phase angle represents a numerical value which is changed by the user pressing the push-buttons. If a button is held depressed, the value should automatically increment or decrement until a certain limit is reached. The zero-crossings of the mains voltage act as reference points for the phase angle control. This reference signal is supplied to the microcontroller by way of a port line with interrupt functions. If the interrupt occurs, a delay has to be created based on the previously established

phase angle. When the delay has elapsed, the triac is fired (triggered). To keep the hardware layout for the power supply etc. within reason, the triac gate receives several short pulses instead of one long trigger signal.

Programming

The next step is to convert the above functional description into symbolic language. For this, the designer has access to several libraries from which the various sub-functions and logic conditions may be fetched and combined into a drawing.

Version 'potentiometer'

The 'symbolised' main program of the potentiometer-controlled version of the dimmer is shown in **Figure 1**. The A-D converter symbol supplies an 8-bit number which is divided by two and limited to minimum and maximum values. The number computed in this way is in the range 0 to 90 and corresponds to a phase angle between 0 % and 100 %. The symbol 'subscheme' (sub-schematic) serves to copy this value to the interrupt routine. The controller initialisation, the starting and scanning of the A-D converter, the division by two and so on are described in the associated library functions and remain 'in the background', that is, invisible to the programmer.

Because the main program has a very simple structure, no logic decisions are required. Consequently, the output from the A-D converter is read, arithmetically modified and then copied to the interrupt routine each time the main program loop is executed. The main program flow, or 'state machine' is therefore not required in this example.

The associated interrupt routine is shown diagrammatically in **Figure 2**. It has to be drawn on a separate sheet and is marked by the 'event' symbol. Here, this symbol was placed in the left-hand area of the worksheet. By double-

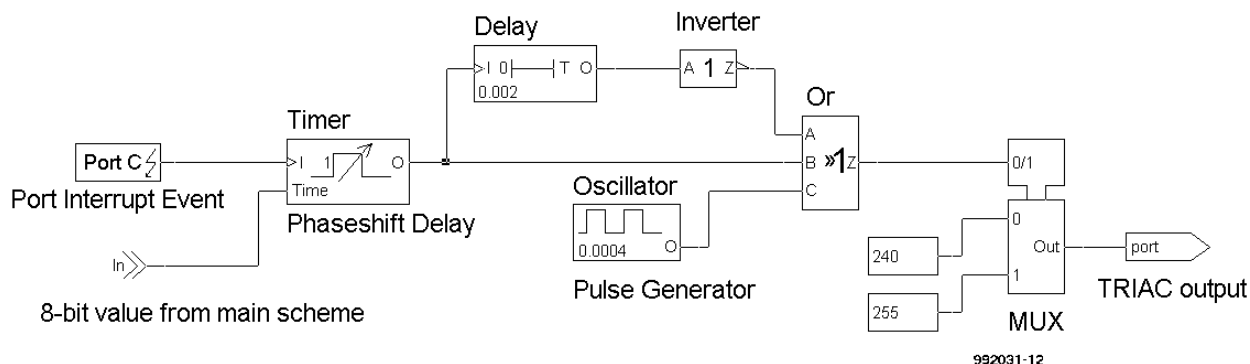


Figure 2. Subroutine 'port interrupt'.

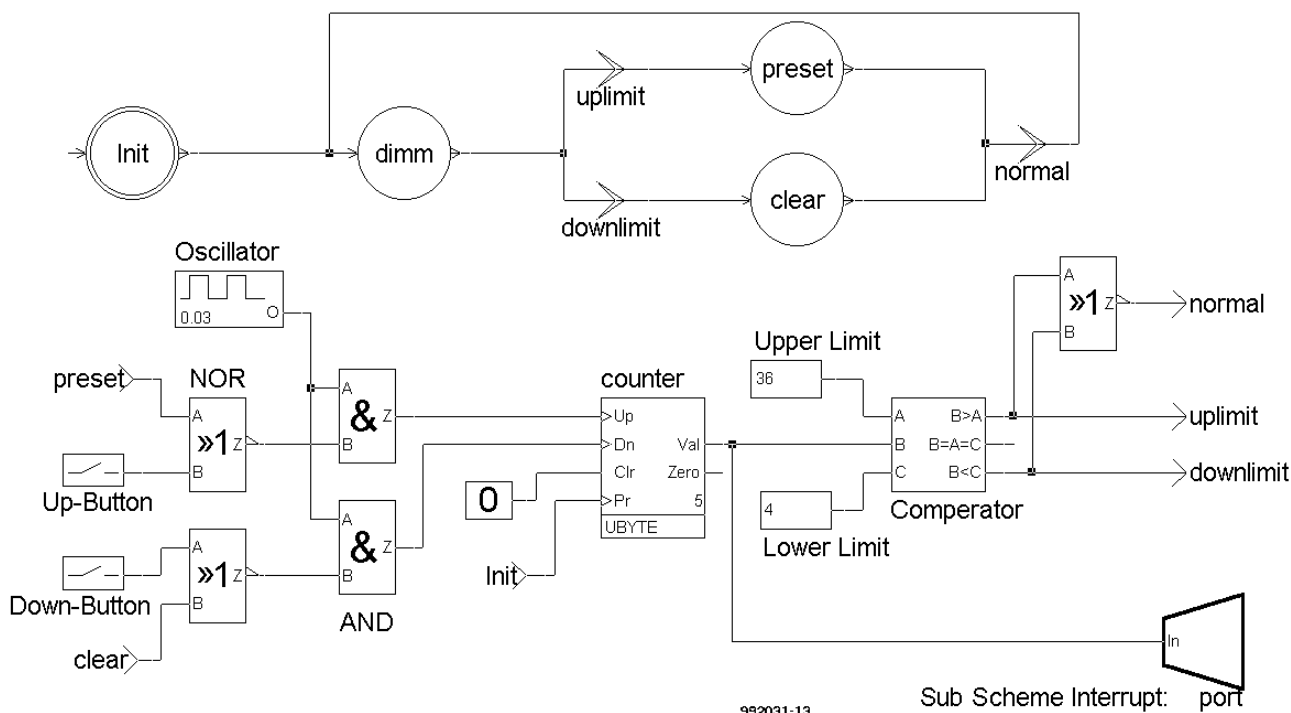


Figure 3. Main program of example version 'B' (dimmer with pushbutton control).

clicking on it, the interrupt type and pin number may be defined. When the interrupt is generated (on a zero-crossing of the mains voltage) a timer is started using the computed value supplied by the main program. Next, the timer generates the desired phase control angle. When the period has elapsed, a burst of five pulses of 0.2 ms each is generated with the aid of an oscillator, a delay, an inverter and a triple OR gate. The function of the multiplexer is to supply the pulse train simultaneously on the four port pins. This is done to boost the drive capacity. The ports are defined by double-clicking

on the 'output' symbol.

At this point the present version of the application is fully described and ready for compiling. Up to then, the user should not have committed himself to a certain microcontroller. The main choice to make is between the ST6 and ST7 families from STmicroelectronics. From these families, you first have to select a 'large' member, because it will be hard to tell, at this point, the amount of resources used up by the target program. During the 'translation', ST-Realizer provides you with a report about possible errors and also supplies warnings if anything ambiguous is encoun-

tered. At the end, a statistic report is created, providing exact information on the required resources.

With this information to hand, the programmer may estimate how much additional functionality may be packed into the microcontroller, or if a smaller controller with less memory is also sufficient for the relevant task. Because the graphics-based program description is not absolutely dependent on the hardware, switching between controllers at a later time is possible without problems and without the need for a new drawing. The only action required in these cases is re-assigning the inputs and outputs by double-clicking on the new I/O pins.

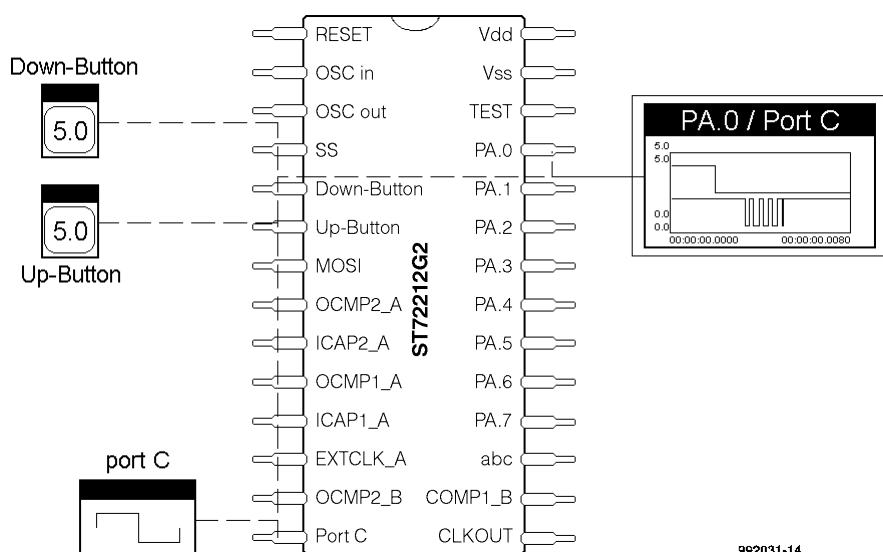


Figure 4. Pin level simulation.

Version 'pushbuttons'

The second version of the application employs two pushbuttons to set the lamp intensity. In the schematic, this function is represented by two symbols of a digital input. As you can see in **Figure 3**, each symbol of a logic input has one output. To be able to understand the operation of the ST-Realizer, it is essential to know the following convention: the output of input symbols represents the logic level at the input pin, and is therefore either '0' or '1'. On this line, pressing the pushbutton causes a 1-0-1 state change which, being applied to a counter input, allows a value to be incremented or decremented. Between the input and the counter sits a NOR gate. This logic component prevents the counter value from

going outside the limits set up to represent the largest and smallest phase control angle. That, of course, requires the user to develop the check on the high and low limits of the counter values. For this purpose, the main program includes a state machine. As shown in Figure 3, this symbolic flowchart consists of states (loops) in which a function is performed, and conditions (the double arrows) which have to be satisfied to move from one state to the next. Here, too, it is essential to understand that an active state name (e.g. 'dim') is represented by a logic '1', while all inactive state names equate to logic '0'. If the active name pops up in other parts of the drawing, then all circuit sections connected to it are also at logic '1'. The change between two states occurs when the name of the condition is set to '1' by a circuit section connected to it. For example, when the counter reaches the upper limit (maximum), the relevant comparator output changes to '1'. This output is identified as 'uplimit'. Consequently, the state machine changes from 'dim' mode to 'preset'. That, in turn, causes the name 'preset' to become active, disabling the 'up-button' input until the timer is decremented again.

Once the right value has been established in this way, it is copied to the interrupt routine. The interrupt routines of version A and B are identical.

Simulation

The user having successfully compiled this version of the program, both programs may be tested using a graphic simulator. The choice is between pin level simulation and a simulation at the schematics level.

The first option displays the microcontroller as a complete integrated circuit (**Figure 4**) whose input pins are driven by signals from a variety of sources — digital or analogue. This type of simulation provides an excellent method of observing and analysing the external behaviour of the component as a whole.

With the second method, the previously mentioned simulators and monitors may be connected to just about any line within the (virtual) circuit. This simulation method is illustrated in **Figure 5**. It is particularly useful for in-depth analysis.

Conclusion

As shown with the aid of an example, the latest version of the ST-Realizer enables time-critical applications to be described in a very short time, without prior knowledge of a controller-specific programming language. Apart from the obvious advantage of the short program development time, the user may also rely on well-tried library functions and so increase the quality of his

program. A nice side-effect of the ST-Realizer is its ability to document the program, whose 'flow' and other salient data appear in condensed form on just a few pages. In conclusion, the latest version of the ST-Realizer represents a valuable addition to traditional programming languages for microcontrollers.

(982093-1)

ST-Realizer may be obtained through your national STMicroelectronics distributor or directly from:

Actum Solutions,
Industriestraat 9a,
1704 AA Heerhugowaard,
The Netherlands.
Tel. (+31) 72 5762555,
fax (+31) 72 5762559,
Email: info@actum.com.
Internet: <http://www.actum.com>

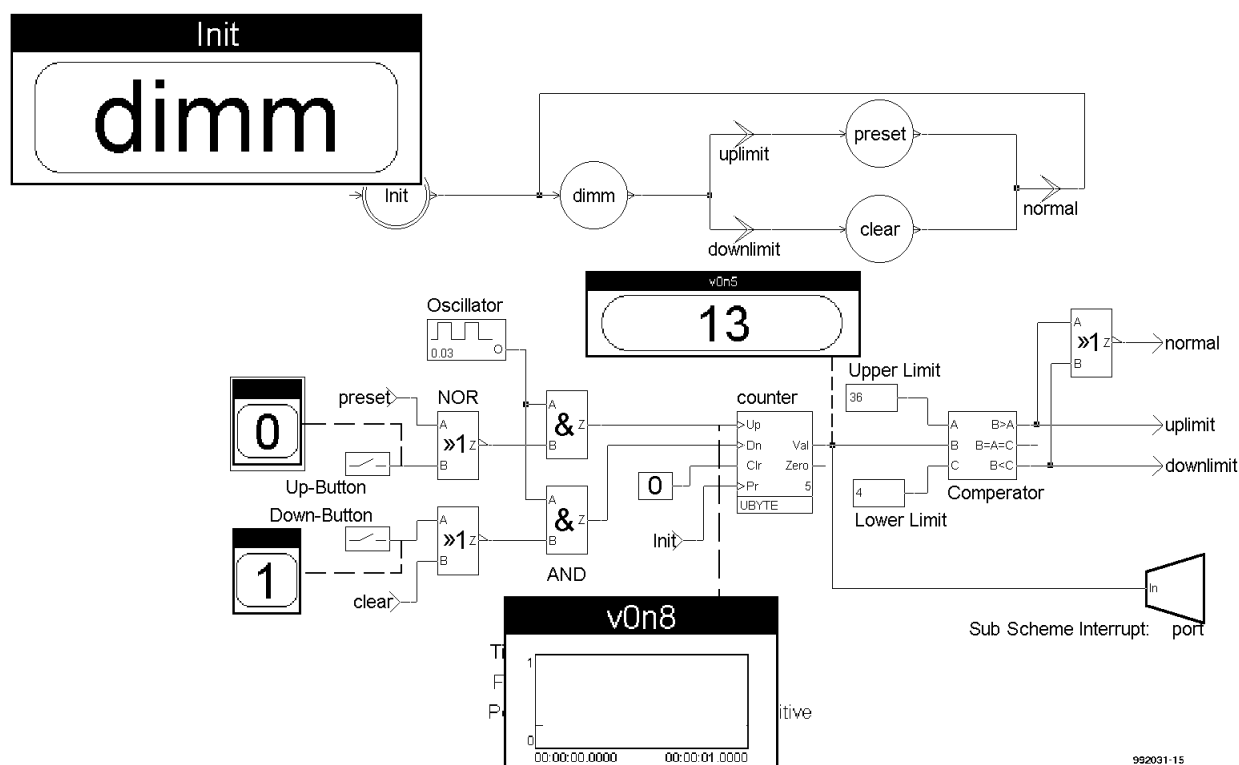


Figure 5. Functional simulation.